A3-Q2: Golf Driving Range

In [5]:
```
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
```

In [6]:
```
# Supplied functions
def Ground(d):
    '''
     h = Ground(d)

     Returns the height (in metres) of the ground at a horizontal distance
     d (metres) from the origin.
    '''
    return 2.*(np.cos(d/4.)-np.sin(d/11.)-1)

def GroundSlope(d):
    '''
     h = GroundSlope(d)

     Returns the slope of the ground at a horizontal distance
     d (metres) from the origin.
    '''
    return 2.*(-1./4*np.sin(d/4) - 1./11*np.cos(d/11.))
```

(a) MyOde

In [7]:
```
def MyOde(f, tspan, y0, h, event=(lambda t,y:1)):
    '''
     t,y = MyOde(f, tspan, y0, h, event=[])

     Numerically solves the initial value problem

        dy(t)/dt = f(t,y)
            y(0) = y0

     using the Modified Euler time-stepping method.

     Input
       f       a Python dynamics function with calling sequence
                  dydt = f(t, y)
       tspan   2-tuple giving the start and end times, [start, end]
       y0      initial state of the system (as a 1D vector)
       h       the time step to use (this is not adaptive time stepping)
       events  an event function with calling sequence
                  val = events(t, y)
               The computation stops as soon as a negative value is
               returned by the event function.

     Output
       t       1D vector holding time stamps
       y       an array that holds one state vector per row (corresponding
               to the time stamps)

       Notes:
           - t and y have the same number of rows.
```

```
        - The first element of t should be tspan[0], and the first
          row of y should be the initial state, y0.

        - If the computation was stopped by the triggering of an event,
          then the last row of t and y should correspond to the
          time that linear interpolation indicates for the zero-crossing
          of the event-function.
    '''

    # Initialize output arrays, tlst and ylst
    t = tspan[0]
    y = deepcopy(y0)

    tlst = []
    ylst = []

    tlst.append(t)
    ylst.append(list(y))

    n = 0
    event_val = event(t,y)

    # fn = f_n
    # fn1 = f_(n+1)
    # ylst[n] = y_n
    # y = y_(n+1)
    while n == 0 or (t <= tspan[1] and event_val >= 0):
        fn = f(t+h, ylst[n])
        for i in range(0,4):
            y[i] = ylst[n][i] + h * fn[i]

        fn1 = f(t+h, y)

        for i in range(0,4):
            y[i] = ylst[n][i] + h / 2 * (fn[i] + fn1[i])

        t += h
        n = n + 1

        tlst.append(t)
        ylst.append(list(y))

        event_val = event(t, y)

    if event_val < 0:
        h1 = event(tlst[-2], ylst[-2])
        portion = h1 / (h1 - event_val)
        tlst[n] = tlst[n-1] + portion * h
        for i in range(0,4):
            ylst[n][i] = ylst[n-1][i] + portion * (ylst[n][i] - ylst[n-1][i])

    return np.array(tlst), np.array(ylst)
```

(b) Dynamics Function: `projectile`

In [8]:
```
def projectile(t, z):
    type(z);
    K = 0.3
```

```
    g = 9.81
    dzdt = [z[2], z[3], -K*z[2], -g-K*z[3]]
    return dzdt
```

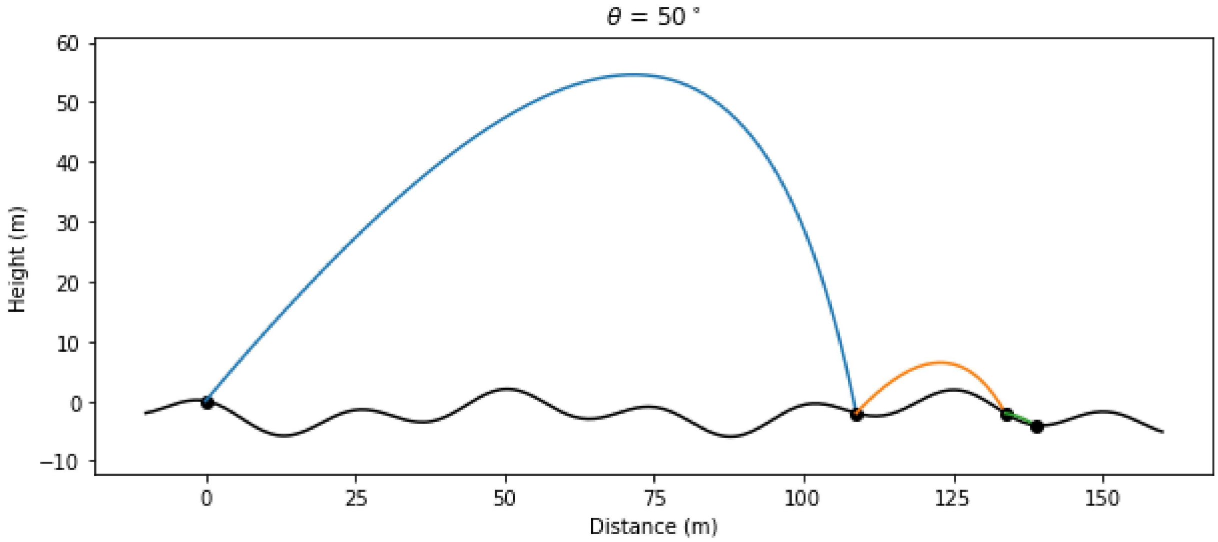
(c) Events Function: `projectile_events`

In [9]:
```
def projectile_events(t, z):

    return z[1] - Ground(z[0])
```

(d) Three flights

In [10]:
```
theta = 50
S = 58
tspan = [0, 30]
h = 0.05
theta_rad = theta/180.*np.pi
yStart = np.array([0, 0, S*np.cos(theta_rad), S*np.sin(theta_rad)])

t_list = []
y_list = []

for i in range(0, 3):
    t,y = MyOde(projectile, tspan, yStart, h, projectile_events)
    slope = GroundSlope(y[-1][0])
    u = [np.cos(slope), np.sin(slope)]
    U = [-u[1], u[0]]
    v = [y[-1][2], y[-1][3]]

    yStart[0] = y[-1][0]
    yStart[1] = y[-1][1]
    # V = 0.85 * (np.dot(v,u)*u - np.dot(v,U)*U)
    yStart[2] = 0.85 * (np.dot(v,u)*u[0] - np.dot(v,U)*U[0])
    yStart[3] = 0.85 * (np.dot(v,u)*u[1] - np.dot(v,U)*U[1])

    tspan = [t[-1], t[-1]+30]

    t_list.append(t)
    y_list.append(y)
```

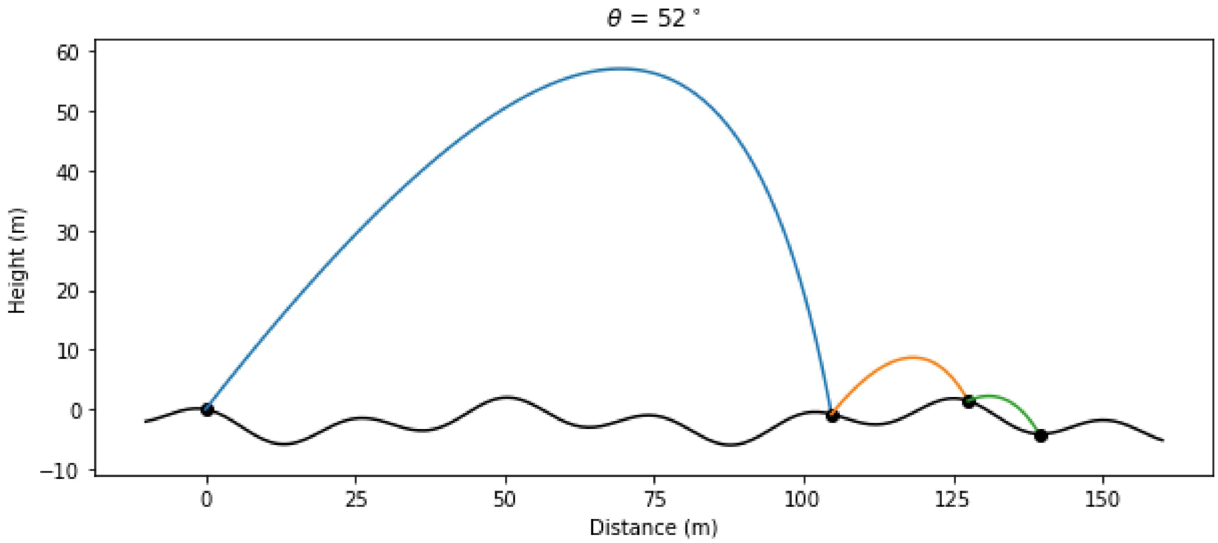
In [11]:
```
# Plot the ground
x = np.linspace(-10, 160, 300)
hills = Ground(x)
plt.figure(figsize=[10,4])
plt.plot(x,hills, 'k')
plt.axis('equal')

plt.plot([0],[0], 'ko')   # Plot initial ball position

for i in range(0,3):
    plt.plot(y_list[i][:,0], y_list[i][:,1])  # Plot ball trajectory
    plt.plot(y_list[i][-1,0], y_list[i][-1,1], 'ko')  # Plot final ball position

plt.title(r'$\theta$ = '+str(theta)+'$^\circ$');
plt.xlabel('Distance (m)')
plt.ylabel('Height (m)');
```



(e) It turns out that theta=52 gives the furthest horizontal distance.

In [12]:
```
theta = 52
S = 58
tspan = [0, 30]
h = 0.05
theta_rad = theta/180.*np.pi
yStart = np.array([0, 0, S*np.cos(theta_rad), S*np.sin(theta_rad)])

t_list = []
y_list = []

for i in range(0, 3):
    t,y = MyOde(projectile, tspan, yStart, h, projectile_events)
    slope = GroundSlope(y[-1][0])
    u = [np.cos(slope), np.sin(slope)]
    U = [-u[1], u[0]]
    v = [y[-1][2], y[-1][3]]

    yStart[0] = y[-1][0]
    yStart[1] = y[-1][1]
    # V = 0.85 * (np.dot(v,u)*u - np.dot(v,U)*U)
    yStart[2] = 0.85 * (np.dot(v,u)*u[0] - np.dot(v,U)*U[0])
    yStart[3] = 0.85 * (np.dot(v,u)*u[1] - np.dot(v,U)*U[1])

    tspan = [t[-1], t[-1]+30]

    t_list.append(t)
    y_list.append(y)

x = np.linspace(-10, 160, 300)
hills = Ground(x)
plt.figure(figsize=[10,4])
plt.plot(x,hills, 'k')
plt.axis('equal')

plt.plot([0],[0], 'ko')   # Plot initial ball position

for i in range(0,3):
    plt.plot(y_list[i][:,0], y_list[i][:,1])  # Plot ball trajectory
    plt.plot(y_list[i][-1,0], y_list[i][-1,1], 'ko')  # Plot final ball position
```

```
plt.title(r'$\theta$ = '+str(theta)+'$^\circ$');
plt.xlabel('Distance (m)')
plt.ylabel('Height (m)');
```



In [ ]:

In [ ]: